

9/PRTS

10/521963

DT01 Rec'd PCT/PTO 21 JAN 2005

**Digital data compression robust relative to transmission noise**

- 5 The invention relates to digital data compression, in particular for multimedia signals (audio, image, video, voice), and the robust transmission of this data on noisy networks, such as wireless and mobile communications networks.

10 To reduce the transmission rate of digital data, it is compressed, seeking at the same time to approach the theoretical maximum that specialists refer to as "signal entropy". To do this, use is often made of statistical codes also termed variable length codes, for example Huffman codes. However, these codes have the drawback of being very sensitive to transmission errors. Inversion of a bit can lead to de-synchronization of the decoder, which results in erroneous decoding of all the data following the position of  
15 the erroneous bit.

Existing solutions for the compression, transmission and decompression of multimedia signals over a network, to which further reference will be made, are based on the hypothesis that a certain quality of data transport service is guaranteed. In other words  
20 they assume that, by relying on the use of correction codes, the transport and link layers will make it possible to achieve a quasi-null residual error rate (i.e. seen by the compression and decompression application). But this hypothesis of quasi-null residual error rate no longer holds true when the channel characteristics vary over time (non-stationary channels), in particular in wireless and mobile networks, and for a realistic  
25 complexity of the channel code. Furthermore, the addition of redundancy by correction codes leads to a reduction of the effective data rate.

There is therefore a need for solutions that are robust relative to transmission noise, i.e. that are little affected by the bit errors induced by this noise on the one hand, and which  
30 facilitate optimal use of the bandwidth (i.e. the capacity) of the network on the other hand.

The present invention proposes some advances in this area.

In one of its aspects, the invention discloses a digital data compression encoder, which includes:

- an input (physical or otherwise) for a first data flow, and a second data flow,
- 5 - an encoding module, matching symbols of the first data flow and code words, wherein, for certain symbols, there exist several words, called redundant, corresponding to the same symbol, and
- a processing module for encoding the symbols of the first data flow based on the match, by selecting among the redundant words, on the basis of at least part of the
- 10 second data flow.

In various other aspects:

- the code words can be of fixed length,
- the processing module includes:
  - 15 . a function to calculate the current multiplexing capacity of the first data flow, based on the coding table, and
  - . a function to extract a multiplexed part from the second data flow, determined on the basis of the current multiplexing capacity, to be carried by said redundant words.
- 20 - the encoder includes a transformation of a binary flow into a multi-valued variable flow, in particular using the transformations described in Table C below.
- as a variant, the encoder includes a transformation of a binary flow into a multi-valued variable flow, in particular using a generalized Euclidian decomposition based on a global variable given by the formula (E9) described below.

25

In a first variant :

- the encoding module includes a coding table and the processing module includes:
  - a function for reading a multiplexing capacity of each current symbol of the first data flow based on the coding table, and
  - 30 - a function for extraction of a part of the second data flow determined from the multiplexing capacity, to be carried by said redundant words,

- the coding table includes for each symbol an associated number of code words equal to a power of 2.

In a second variant :

- 5 - the encoding module includes a binary encoding tree containing, for each symbol of the first data flow, a first code word part, of variable length and shorter than a maximum length, and the processing module includes:
  - a function to calculate the multiplexing capacity for each current symbol of the first data flow based on the first code word part of each symbol,
  - 10 - a function to extract a part of the second data flow determined from the multiplexing capacity, to be carried by said redundant words.
- as a variant, each symbol comprises a sequence of symbols.

In a third variant :

- 15 - each symbol includes a sequence of symbols, and the encoding module includes an arithmetic encoder capable of calculating, for a sequence of symbols of the first data flow, a first code word part of variable length and shorter than a maximum length; the processing module includes:
  - a function to calculate the multiplexing capacity for each current symbol of the first data flow based on the first code word part of each symbol,
  - 20 - a function to extract a part of the second data flow determined from the multiplexing capacity for each symbol, to be carried by said redundant words.

25 In the second and third variant, said part of the second data flow is concatenated to the first code word part up to the maximum code word length.

In a general manner :

- the second data flow is pre-encoded.
- the rest of the second data flow is concatenated to the transmitted data.

30

The invention also discloses a decoder capable of performing the inverse or reciprocal operations relative to those of the encoder in its different aspects.

The invention also discloses a digital data compression method that includes the following steps:

- a. establishing a match between symbols of the first data flow and code words,
  - 5 wherein, for certain symbols, there exist several words, termed redundant, corresponding to the same symbol, and
  - b. encoding the symbols of a first data flow based on step a., by selecting among the redundant words, on the basis of at least part of a second data flow.
- 10 This method can incorporate other aspects of encoding.

In addition, the invention also discloses a digital data decompression process, including steps reciprocal to those of the compression process.

- 15 Other features and advantages of the invention will become apparent upon examination of the following detailed description together with the attached drawings in which :
- Figure 1 is a flow diagram illustrating a code creation method,
  - Figure 2 is a diagram giving an overview of the encoding method in its principal variant,
  - 20 - Figure 2A illustrates a simplified example of a multiplexed code, with four elements,
  - Figure 3 illustrates a first embodiment of a detailed encoding method,
  - Figure 4 illustrates an alternative embodiment of the method in Figure 3,
  - Figure 5 illustrates an example of the creation of a storage capacity by assigning several code words to a symbol, a data flow  $q$  being capable of being jointly stored, and
  - 25 - Figure 6 or Table C illustrates transformations used in an example where the parameter  $f_v$  is equal to 5,
  - Figure 7 illustrates another simplified example of a four-element multiplexed code,
  - Figure 8 illustrates a general encoding method for Figures 3 and 4,
  - Figure 9 illustrates a second embodiment of a detailed encoding method,
  - 30 - Figure 10 illustrates a results table of a first variant of the second embodiment of the encoding method according to Figure 9,

- Figure 11 illustrates an overview of a second variant of the second embodiment of the encoding method according to Figure 9,
- Figure 12 illustrates an overview of a third variant of the second embodiment of the encoding method according to Figure 9.

5

In addition:

- Appendix 1 contains expressions used in the present description, and
- Appendix 2 contains natural language algorithms used in the present description.

10 The drawings and the appendices essentially include elements that are certain in character. They will therefore serve not only to aid understanding of the description, but will also contribute to the definition of the invention, as applicable.

15 In a general manner, a compression system for multimedia signals (image, video, audio, voice) uses statistical codes also termed variable length codes. These make it possible to obtain data rates approaching what the specialists refer to as "signal entropy". The codes most widely used in existing systems (particularly in the standards) are Huffman codes which have been described in the following paper: D.A. Huffman : "A method for the construction of minimum redundancy codes", Proc. IRE, 40 (1951), p.1098-1101.

20

More recently, there has been renewed interest in arithmetic codes owing to their increased performance in compression terms. In effect these codes make it possible to decouple the encoding method from the supposed source model. This facilitates the use of higher-order statistical models. These arithmetic codes have been described in research papers such as

25

- J.J. Rissanen "Generalized kraft inequality and arithmetic", IBM J.Res. Develop., 20:198-203, May 1976

- J.J. Rissanen, "Arithmetic coding as number representations", Acta Polytech. Scand. Math., 31:44-51, December 1979

30

and in American patents US 4,286,256, US 4,467,317, US 4,652,856.

Until recently, the design of compression systems was undertaken assuming a guaranteed quality of transport service. It was assumed in fact that the lower layers of the OSI model incorporate error correction codes ensuring a quasi-null residual error rate as seen by the application.

5

Variable length codes could therefore be widely used despite their considerable sensitivity to transmission noise. Any error in the binary train can cause desynchronization of the decoder and therefore propagation of errors to the decoded information stream.

10

To mitigate this propagation problem, the first-generation standards (H.261, H.263, MPEG-1, MPEG-2) incorporated synchronization markers into the syntax of the transmitted binary train. These are long code words (16 or 22 bits composed of a string of 15 or 21 bits set to '1' followed by a '0') which cannot be emulated by errors occurring in the other code words and which can therefore be recognized by the decoder with a probability close to '1'.

15

This leads to the binary train being structured in packets delimited by these synchronization markers. This enables the propagation of errors to be confined within a packet. However if an error occurs at the start of the packet the rest of the packet may be lost. Moreover, the frequency of these synchronization markers must be restricted to avoid undue loss of compression efficiency.

20

This hypothesis of quasi-null residual error rate no longer holds true in wireless and mobile networks in which the channel characteristics vary over time (non-stationary channels). This residual error rate seen by the source signal decoder is often far from negligible.

25

The new standards (H.263+ and MPEG-4) then adopted reversible variable length codes (RVLC). A particular feature of these codes is that they can be decoded from the first to the last bit in a packet, and inversely from the last to the first bit in the packet.

30

If an error has occurred in the middle of a packet, this code symmetry makes it possible to confine the error propagation to a segment in the middle of the packet instead of propagating it to the end of the packet delimited by a synchronization marker. However, the code symmetry results in a loss of compression efficiency in the order of 10% compared with a Huffman code. In addition, reversible variable length codes do not completely overcome the problem of error propagation: if an error occurs at the start and end of a packet, the whole packet is liable to be erroneous.

The design of codes that are both efficient in compression terms (i.e. approaching entropy of the source) while at the same time being robust relative to transmission noise, is therefore an important goal, in particular for future multimedia (image, video, audio, voice) mobile communications systems. New standards are being developed for these systems both within the ITTU (International Telecommunication Union) and ISO (International Organization for Standardization).

Although standards play a predominant role in the telecommunications sector, such a family of codes could also find applications in niche markets calling for proprietary solutions.

In a general manner, the digital data compression methods proposed below are implemented by a digital data compression encoder according to the invention including an encoding module and a processing module.

More specifically, an encoding module establishes a match between symbols in a first data flow and code words, wherein for certain symbols there exist several words, termed redundant, corresponding to the same symbol. In a general manner, an encoding module can be any form of storage presenting the match defined above, any form of representation of this match, or any coding function calculating this match. Thus, by way of example only, the encoding module can be an encoding table, an encoding tree or an arithmetic encoder depending on the compression methods detailed below.

The processing module implements the stages of the data compression methods based on the first data flow and a second data flow and the encoding module. The processing module includes functions capable of performing certain steps in the processes and can include a transformation of a binary flow into a multi-valued variable flow. These functions include a function for calculating or reading a multiplexing capacity associated with the first data flow, and a function for extracting part of the second data flow. These functions will be developed more particularly in the rest of the description. In a symmetrical manner, the data decompression methods are implemented by a decoder according to the invention.

In a general manner, the method described below involves creating fixed length codes for higher priority source data (flow  $s_H$ ), by assigning several code words to each possible representation of this source.

Thus, to transmit a symbol, it is possible to choose from the different possible representations of the latter. This choice, which is a multi-valued variable, defines a storage capacity that will be capable of being used to jointly transmit other data (Cf. example in Figure 5 detailed below). These are data of lesser importance, represented by a flow denoted  $s_L$ , which will be represented via multiple representation of the symbols.

Figure 1 shows the method of creating these codes, more precisely the method of creating multiplexed codes.

At step 100, the higher priority source,  $s_H$ , takes its values from an alphabet of  $\Omega$  elements, which can be defined by formula (E1) attached. It is assumed that the probability law  $\mu$  of occurrence of the symbols in this alphabet is known. The notation  $\mu_i$  represents the probability associated with symbol  $a_i$  in the alphabet of the source  $s_H$ , as represented by formula (E2).

The method of creating codes illustrated in Figure 1 can then be broken down into 2 main steps:



- For each symbol  $a_i$ , selection of the number  $N_i$  of code words assigned to this symbol,
- Assignment of code words to the symbols.

5 The first step involves the selection of code parameters  $c$  and  $(N_i)$  broken down into different steps 120, 130, 140 and 150.

At step 120-1, a code word length parameter  $c$ , in number of bits, is chosen. In a first embodiment (in reference to Figures 2A and 5), the parameter  $c$  takes the value  $c = 4$ . In another embodiment (in reference to Figures 2B, 7 and 10), the parameter  $c$  takes the value 3. This defines  $2^c$  code words at step 120-2, to be allocated between the symbols of alphabet A.

Depending on the possible values of the probability  $\mu_i$  at step 130-1, the alphabet A symbol set is partitioned into two subsets  $A_m$  and  $A_M$  respectively at step 130-2 and at step 130-3. The first is the symbol set  $a_i$  of which the probability  $\mu_i$  is less than or equal to  $1/2^c$ , the second is its counterpart in A. The cardinals of these sets are respectively denoted  $\Omega_m$ , and  $\Omega_M$ .

At step 140, the probability law  $\overline{\mu}$  is then calculated on the symbols of  $A_M$ . It is given by formula (E3).

At step 150-1, the number of code words per symbol is then chosen so as to verify approximately formula (E4), under the constraint of formula (E5) for the subset  $A_M$ . A conventional optimization algorithm may be used for this purpose. At step 150-2, the number of code words per symbol is determined for the symbol set of alphabet A.

In a variant of the code creation method including steps 120, 130, 140 and 150, a further step is added after step 150, and steps 120-1 and 150-1 respectively take the following form.

Let  $f_1 = 2, f_2 = 3, f_3 = 5 \dots, f_v$ , be  $v$  first prime numbers. In addition to parameter  $c$  at step 120-1, a prime number  $f_v$  is also chosen from these prime numbers.

At step 150-1, the procedure is the same, but with the addition of a further constraint at step 150-11 on the choice of the number of code words  $N_i$  associated with each symbol: the decomposition into prime factors of all the  $N_i$  must not contain a prime factor greater than  $f_v$ .

After step 150-1, the decomposition of each  $N_i$  into prime factors is then carried out, and for any  $N_i$  the number of times that each prime factor  $f_j$ , with  $1 \leq j \leq v$  occurs in this decomposition is calculated. This number is denoted  $\alpha_{i,j}$ , where  $i$  denotes the symbol  $a_i$  considered and  $j$  denotes the prime number  $f_j$  considered. The correspondence between  $a_i$  and the numbers  $\alpha_{i,1} \dots \alpha_{i,v}$  can be stored in a table termed the "alpha table".

The additional step entails allocating code words to the symbols. This step is broken down into different steps 160 and 180 described below.

At step 160, binary labeling (0000, 0001,...) of the symbols arranged, by way of example only, in a lexicographic sequence is carried out. This makes it possible to assign  $N_i$  code words to the different symbols, the parameter  $N_i$  being an integer determined at the preceding step (150-2). The set of code words thus associated with a symbol  $a_i$  is referred to as the  $a_i$  equivalence class, and denoted  $C_i$  at step 180.

A value between 0 and  $N_i - 1$ , here referred to as a state, is then assigned to the code words of each equivalence class. This value identifies the code word within the equivalence class.

Thus, each code word  $c_i$  is associated with a symbol  $a_i$  and a state variable between 0 and  $N_i - 1$ , as illustrated by the expression (E6).

An example of a code thus constructed is given in Table A, Figure 2A for  $f_v = 5$  and  $c = 4$ . For each symbol  $a_i$  in column 13-A, Table A includes a column 11-A including the classes  $C_i$ , a column 12-A including the code words assigned to each class, a column 14-A including the numbers  $N_i$  of code words per class, a column 15-A including the

probabilities  $\mu_i$  associated with each class, and a column 16-A including a state  $q$  of each code word in a class. In this example, the symbols  $a_i$  include 4 elements of the alphabet.

- 5 A variant of the code constructed is given in Table B, Figure 2B for  $f_v = 2$  and  $c = 3$ . Table B defines a binary multiplexed code, i.e. a multiplexed code such that for any symbol  $a_i$  the cardinal  $N_i$  of the associated equivalence class is an integer power of 2. If we denote this power  $1_i$ , the cardinal  $N_i$  of the equivalence class verifies E(11). For each symbol  $a_i$  in column 13-B, Table B includes a column 11-B including the classes
- 10  $C_i$ , a column 12-B including the code words assigned to each class, a column 14-B including the numbers  $N_i$  of code words per class. In this example, the state variable  $q$  does not exist but other elements are present and their function will be better understood by reading the description: a column 15-B includes the probabilities  $\mu_i$  each assigned to a class, a column 18-B includes a number  $D_i$  of bits capable of being stored,  $D_i$  being
- 15 associated with each class, a column 17-B includes a set  $\overline{U_i}$  of  $D_i$  bits for each code word in a class. In this example, the symbols  $a_i$  include 4 elements of the alphabet with  $i$  taking the values between 1 and 4.

- The condition (E11) on the cardinals of the equivalence classes leads to the fact that the
- 20 choice of a code word within an equivalence class makes it possible to store a whole number of bits equal to the base 2 logarithm of the cardinal of the equivalence class. This number of bits  $D_i$  can therefore be written according to (E12).  $D_i$  represents a multiplexing capacity of a given symbol. Tables A and B are encoding tables, also referred to as multiplexed code word tables or multiplexed code table.

25

- In a third alternative embodiment, a binary multiplexed code can also be constructed from a binary encoding tree associated with a code word prefix, this prefix being of variable length and shorter than a maximum length being the height of the tree, such as the Huffman code presented in Figure 7. The relationship "lower" is to be understood to
- 30 mean "lower or equal". The tree is first divided into two branches respectively taking the value 1 and the value 0. In a recurrent manner, each branch of the tree is divided into two branches respectively taking the value 0 and the value 1. The parameter  $c$  is given

- by the height of the tree, i.e. by the length of the longest prefix (in the example  $c = 3$ ) of the tree. An equivalence class  $C_i$  is defined by a set of fixed-length code words each having a common first part, termed common prefix. This common prefix is the part of the code word with variable length used to represent the symbol  $a_i$  and denoted in Figure 7 by the path formed by the successive branches shown as a solid line. For certain solid-line paths representing a symbol  $a_i$ , there remain a certain number of dotted-line paths each representing a second part of the code word termed suffix. The suffix of a code word defines the bit set  $U_i$  as indicated by an arrow in the drawing.
- Thus, the encoding tree is defined as an encoding module, matching the symbols of the priority data flow and code words, wherein for certain symbols there exist several words, termed redundant, corresponding to the same symbol. These code words are of fixed length and include a first and a second code word part, for example a prefix and a suffix of variable lengths. In a variant, the first and the second code word part can correspond respectively to the suffix and prefix of the code word. More generally, a code word can include several code parts.

Figures 2 and 3 illustrate the coding method, also termed encoding.

- The encoding method is broken down as follows:

- steps 1: the data flow of lower importance  $S_L$  at step 1-1 is encoded in a binary sequence  $b = (b_1, b_2, \dots, b_{KB})$  using a reversible encoding at step 1-2. For this purpose, a Huffman type reversible encoder or arithmetic encoder (non restrictive) can be used. This gives rise to the generation of a sequence of bits, denoted  $b$  at step 1-3.
- steps 2 : from the sequence of symbols  $s_1, s_2, \dots, s_{KH}$  of the flow  $s_H$  at step 2-1 and reading the table of multiplexed code words at step 2-2, the associated values  $n_1, n_2, \dots, n_{KH}$  are derived at step 2-3. For example, for a symbol  $s_1$  corresponding to the symbol  $a_i$  in a table of multiplexed code words,  $n_1$  takes the value of  $N_i$  corresponding to the representation of the symbol  $a_i$ .

- steps 3 : the value  $\Lambda$  is derived, here using formula (E7) at step 3-1. The number  $K_B$  of bits which it will be possible to store using the intrinsic redundancy of the multiplexed codes is calculated using formula (E8) at step 3-2. This number  $K'_B$  of bits represents the multiplexing capacity of the flow  $s_H$  for this encoding method.

5

- steps 4 : on condition that  $K_B < K'_B$  at step 4-1,  $K'_B$  bits of flow  $b$ , for example the last  $K'_B$  bits of flow  $b$  (steps 4-2, 4-3) are used to calculate (step 4-4) a long integer  $\gamma$  (step 4-5), given here by formula (E9). This corresponds to the transformation of the last  $K'_B$  bits of the binary flow into a global variable. If the condition  $K_B < K'_B$  is not verified at

10

step 4-1, the process resumes at step 2-1 by reducing the value  $K_H$  (step 4-11).

- steps 5 : the value  $\gamma$  can then be used to calculate the states  $q_t$ ,  $1 \leq t \leq K_H$  (step 5-2), using for example a generalized Euclidian decomposition method (step 5-1), as illustrated in the attached algorithm (A1). This involves generating a flow of states  $q_t$ ,  $q_t$  being a multi-valued variable.

15

- steps 6 : for any  $t$  such that  $1 \leq t \leq K_H$ , knowing the symbol  $s_t$  and the state  $q_t$  calculated at the preceding step enables the code word to be chosen from the table of multiplexed code words (step 6-1). The multiplexed flow  $m$  is obtained including the code words  $m_1$  to  $m_{K_H}$  (step 6-2).

20

- steps 7 : the  $K_B - K'_B$  bits of the flow of lower importance (step 7-1) are then concatenated to the sequence of multiplexed code words previously constructed (step 7-2) to form the transmitted flow (step 7-3).

25

At least step 3 is implemented by the calculation function of the processing module. Steps 4, 5 and 6 at least are implemented by the extraction function of the processing module.

30

In a general manner, for an encoding method, an associated decoding process is carried out by performing operations that are the reverse of those of the encoding method.

A variant of the encoding method is illustrated in Figure 4 and avoids performing calculations on long integers. The variant of the encoding method is broken down as follows:

5 - steps 21 : these correspond to steps 1 in Figure 3.

- steps 22 : these correspond to steps 2 in Figure 3.

10 - steps 23: the total number of times that each prime factor  $f_j$  appears in the set of decompositions into factors of the sequence of variables  $N_t$  is then determined from the so-called "alpha" table (steps 23-1 and 23-2). This number is denoted  $d_j$  below, and represents the number of  $f_j$ -valued variables that can be multiplexed with the flow  $s_H$ . Thus, for each prime factor  $f_j$ ,  $d_j$  represents the sum of  $\alpha_{t,j}$  in the sequence of variables  $n_t$ .

15

- steps 24 : the transformations that will be used to transform the binary train into these  $f_j$ -valued variables are then chosen (step 24-1). These transformations depend on the value of  $f_v$  chosen. The transformations used for  $f_v=5$  are presented in Table C of Figure 5.

20

They are presented in the form illustrated in the attached formulas (E10).

Thus, each transformation  $T_z$  takes  $u_{Tz}$  bits at the input (denoted below  $u_T$  by simplification for a given  $z$ ) and transforms them respectively into  $v_{T,1}, v_{T,2}, \dots, v_{T,v}$  variables 2, 3, 5, . . . ,  $f_v$ -valued. In the example in Table C for  $f_v = 5$ , each transformation  $T_z$  in column 31 takes  $u_T$  bits in column 32 as input and transforms them respectively into  $v_{T,1}, v_{T,2}, v_{T,3}$  variables 2, 3, 5-valued in columns 33, 34, 35. The required number of variables of each type is known: for each type of variable  $f_j$ , it is  $d_j$  (Cf. step 23-2).

30

The attached algorithm A2 can be used to calculate the number  $g_{Tz}$  of times that the transformation  $T_z$  must be used (step 24-2), for a variable  $z$  ranging from 0 to  $z_{\max}$ . (It

is assumed that the transformations are arranged in descending order of relevance in the table).

- step 25 :  $K_B$ , the number of multiplexed bits, is calculated by obtaining the product of the number of times  $g_{Tz}$  that a transformation  $Tz$  must be used and the number of bits  $u_T$  at the input of the transformation and adding these products together for all the transformations  $z$  used. This number  $K_B$  of bits represents the multiplexing capacity of the flow  $s_H$  for this encoding method. This step 25 corresponds to steps 3 in Figure 3. The following steps 26-1, 26-2, 26-3, and 26-11 correspond to steps 4-1, 4-2, 4-3 and 4-11 in Figure 3.

- steps 27 : having chosen the number of transformations of each type to be used, they are applied to the end of the binary flow  $b$  (step 27-1).

For each transformation  $Tz$ , the  $u_T$  input bits are seen as the binary representation of an integer  $e$ .

This integer is then decomposed into several  $f_j$ -valued variables, as indicated in the formulas (E10). These variables are denoted  $e_{r,j}$ , where :

$j$  indicates that the value obtained is the representation of an  $f_j$ -valued variable, and  $r$  indicates the number of the  $f_j$ -valued variable.

Values of  $e_{r,j}$  can be obtained from  $e$  using the method of algorithm A3. This algorithm is reiterated a number  $g_{Tz}$  of times for each transformation  $Tz$ .

On completion of this step 27-1, the results obtained are presented in the form of formulas (E10) and are concatenated so as to obtain  $v$  sequences of available variables (step 27-2):

- the first, denoted  $F_1$ , is a sequence with a length  $d_1$  of 2-valued variables (bits),  
 - the  $j$ -th, denoted  $F_j$ , is a sequence with a length  $d_j$  of  $f_j$ -valued variables. Position pointers, denoted  $t_j$ , are assigned to the sequences, and are initially positioned at the start of each sequence.

- steps 28 : the flow of states (step 28-1) is calculated from these variables, the result of which is (step 28-2) :

$$q = (q_1, q_2, \dots, q_{K_H}).$$

5

This calculation may be performed by proceeding as follows:

- for any  $t$  such that  $1 \leq t \leq K_H$ , and thus for each symbol  $s_t$ , the decomposition into prime factors of  $n_t$  makes it possible to determine the number  $\alpha_{ij}$  of variables of each type (2-valued, ...,  $f_j$ -valued, ...,  $f_v$ -valued,  $j$  ranging from 1 to  $v$ ). Each sequence  $F_j$  previously obtained is divided into  $K_H$  successive segments comprising  $\alpha_{ij}$  bits for  $t$  ranging from 1 to  $K_H$ . The process is reiterated for  $j$  ranging from 1 to  $v$ . Each  $n_t$ -valued variable ( $q_t$ ) is obtained by the reciprocal process of iterative Euclidian decomposition, applied to the segments  $F_{ij}$  of  $f_j$ -valued variables. An example of implementation of this process is described by algorithm A4. It will be noted that at the end of these steps 28,  
15 all the variables of the flows  $F_j$  have been used.

- steps 29 : for any  $t$  such that  $1 \leq t \leq K_H$ , knowing the symbol  $s_t$  and the state  $q_t$  calculated at the preceding step enables the code word to be chosen from the table of multiplexed code words (step 29-1). The multiplexed flow  $m$  is then obtained (step 29-2).  
20

- steps 30 : the  $K_H - K'_H$  bits of the flow of lower importance (step 30-1) are then concatenated to the sequence of multiplexed code words previously evaluated (step 30-2). The transmitted flow is obtained (step 30-3).  
25

At least step 25 is implemented by the calculation function of the processing module. At least step 27 is implemented by the extraction function of the processing module.

The encoding methods presented in reference to Figures 2, 3 and 4 can be generalized according to the encoding process in Figure 8:  
30



- based on the table of multiplexed code words and the sequence of symbols  $s_H$ , the associated values  $n_1, n_2, \dots, n_{KH}$  are calculated so as to calculate the multiplexing capacity  $K'_B$  of the sequence of symbols  $s_H$  at step I.
- the pre-encoded flow  $b$  is divided into two parts  $b'$  and  $b''$  at step II in relation to the multiplexing capacity  $K'_B$ ,
- the part  $b'$  of the flow is transformed into a series of states  $q$  using the values  $n_1, n_2, \dots, n_{KH}$  at step V,
- based on this series of states  $q$  and the table of multiplexed code words, the multiplexed code words are selected at step VII,
- these code words are assembled to form a multiplexed flow  $m$  at step VIII,
- the concatenated part  $b''$  of the flow  $b$  is concatenated with the multiplexed flow  $m$  at step IX.

At least step I is implemented by the calculation function of the processing module. At least step II is implemented by the extraction function of the processing module.

An example of the creation of a storage capacity according to the invention is illustrated in figure 5. Thus, for each symbol  $s_i$  in the data flow  $s_H$  a corresponding class  $C_i$  and the associated code words  $C_{i,q}$  are assigned relative to an encoding table. Each state  $q_i$  of a data flow  $q$  can be jointly stored after selection of the code word  $c_{i,q_i}$  from the table of multiplexed code words.

In the case of conversion of the lower priority binary train, the variant of the encoding method with  $f_v = 2$  can be used. Another variant described below in reference to Figure 9 can advantageously be used. Table B in Figure 2B is used in this example.

The part of the conversion process for the binary train therefore consists of the following steps:

- step 40 : corresponds to steps 1 in Figure 3.

- step 42 : The associated value  $D_t$  is derived by reading the symbol  $s_t$  and reading the table of binary multiplexed codes. This value  $D_t$  corresponds to the number of bits that can be jointly encoded with  $s_t$ .  $D_t$  is a multiplexing capacity of a given symbol.

5 - step 44 : the next  $D_t$  bits of the pre-encoded binary train  $b$  are read. It will be noted that the binary train is read progressively in relation to a positioning pointer. These next  $D_t$  bits are denoted  $\overline{u_t}$  and play the same role as the states ( $q_t$ ).

- step 46 : the code word  $c_{st, \overline{u_t}}$  is selected from the table of binary multiplexed codes as a function of the symbol  $s_t$  and the bits  $\overline{u_t}$ , the table being indexed by  $a_t$  and  $\overline{U_t}$ . This code word is transmitted on the channel,

- step 48 : for each symbol  $s_t$  of the flow  $S_H$ , with  $t$  in the range from 1 to  $K_H$  steps 42 to 46 are performed.

15

At least step 42 is implemented by the calculation function of the processing module. At least step 44 is implemented by the extraction function of the processing module.

By way of an example of application of the process in Figure 9, the highest priority sequence to be transmitted  $s_H = a_H a_2 a_2 a_3 a_2 a_1 a_2 a_4 a_1 a_2$ , of length  $K_H = 10$ , and the low priority pre-encoded binary train  $b = 01010101010$ , are considered. The number of bits that can be multiplexed with each representation of  $s_H$  is given, for  $t$  ranging from 1 to  $K_H$ , by  $(D_t) = (1, 2, 2, 0, 2, 1, 2, 0, 1, 2)$ . The number of bits  $d_t$  in the binary train  $b$  is read progressively for  $t$  ranging from 1 to  $K_H$  so as to obtain the sequences  $\overline{u_t}$  of bits  $(\overline{u_1}, \dots, \overline{u_{K_H}}) = (0, 10, 10, \emptyset, 10, 1, 01, \emptyset, 0, 10)$ . Then, for any  $t$ , the combination  $(a_t, \overline{u_t})$  indexes a code word in the binary multiplexed code table. The binary train effectively transmitted is 000 100 100 110 100 001 011 111 000 100.

As a variant, the process in Figure 9 can also use the encoding tree. In this variant, steps 42 to 46 take the following form :

30

- step 42 : by reading the symbol  $s_t$  and reading the binary encoding tree, the code word prefix for the symbol  $s_t$  is obtained. The number of bits in this prefix is used to derive the number of bits  $D_t$  that can be jointly encoded with  $s_t$  to form a sequence of bits having a total length equal to the height of the encoding tree.  $D_t$  is a multiplexing capacity of a given symbol.

- step 44 : the next  $D_t$  bits of the pre-encoded binary train  $b$  are read. It will be noted that the binary train is read progressively in relation to a positioning pointer. These next  $D_t$  bits are denoted  $\overline{u_t}$ .

- step 46 : the code word transmitted on the channel results from the concatenation of the code word prefix for the symbol  $s_t$  and the bits  $\overline{u_t}$  of the binary train  $b$ . Thus, utilization of the binary train  $b$  enables a choice to be made between the possible code words shown as dotted lines on the encoding tree in Figure 7 for a given symbol.

At least step 44 is implemented by the extraction function of the processing module.

By way of example, using the sequence  $s_H$  and the binary train  $b$  indicated previously in the case of Figure 10 and the encoding tree in Figure 7 to determine the code word prefixes, the code word flow  $m_t$  is obtained by concatenation of the prefixes and suffixes  $\overline{u_t}$ .

In a general manner, the encoding tree makes it possible to define a code word prefix for each symbol in the flow  $s_H$ , which is equivalent to defining several possible code words for certain symbols. The choice between these possible code words will be made once the binary code to determine the code word suffix has been read and the code word has been formed by concatenation of the prefix and suffix. Calculation of the sum of the set of  $D_t$  associated with the symbols forming the flow  $s_H$  makes it possible to determine the multiplexing capacity of the flow  $s_H$ .

Other variants of the encoding method are illustrated below in reference to Figures 11 and 12.

It may be useful to consider creating a multiplexed code not on the alphabet but on a "product alphabet". The term "product alphabet", refers to an alphabet composed not of symbols but sequences of symbols. In the example in Figure 11, the source  $s_H$  comprises  $K$  symbols. It is converted into a  $C$ -uplet source denoted  $H$  (of length  $K/C$ ), these  $C$ -uplets being designated  $H_1, H_2, H_3, \dots, H_{K/C}$  and respectively numbered 51, 52, 53 and 55. Any  $C$ -uplet of symbols has the probability of occurrence (E13). It is used to derive the calculation of the probability distribution  $\mu_H$  associated with the  $C$ -uplets. The binary encoding tree (termed "product binary encoding tree") is created by considering, for each  $C$ -uplet, the probabilities of occurrence of each sequence of length  $C$  given in (E13). The code word prefix associated with a sequence of symbols is read from the encoding tree.

According to Figure 11, for each sequence of symbols, the multiplexing function comprises a certain number of functions performing the steps corresponding to the variant process in Figure 9. At each step, the "symbol" is replaced by a "sequence of symbols". Thus, the encoding method using an encoding tree is applied directly to the  $C$ -uplet representations of the source  $H$ .

If alphabet  $A$  is too large to be able to use an encoding tree, it is also possible to replace "the product encoding tree" by an arithmetic code as illustrated in Figure 12. Thus, the source  $s_H$  is divided into  $C$ -uplets, which leads to a number of  $C$ -uplets equal to  $K/C$ . These  $C$ -uplets can be relatively long and are encoded by independent arithmetic (non restrictive) encoders. In the example in Figure 12, each  $C$ -uplet is encoded by a separate arithmetic encoder 71, 72, 73 and 75. The output of these arithmetic encoders consists of sequences  $H_1, H_2, H_{K/C}$  of variable length bits numbered 81-1, 82-1, 85-1. The length  $c$  of the code words corresponds to the longest possible sequence of bits  $H_t$  at the output of the arithmetic encoders. Each sequence of bits is then seen as a code word prefix. For each prefix of length strictly shorter than the length  $c$ , there exist several code words of length  $c$  corresponding to the same symbol.

The formation of a code word is the same as in the "product alphabet" variant. Thus, the encoded binary flow  $b$  numbered 90 is read progressively to form the suffixes 81-2, 82-2, 85-2 thereby complementing the prefixes 81-1, 82-1, 85-1 and forming the multiplexed code words. If the number  $K/C$  is not an integer, the last  $C' = K - C \lfloor K/C \rfloor$  symbols form a  $C'$ -uplet which is encoded arithmetically.

As indicated for Figure 9, the number of bits in a prefix is used to derive the number of bits  $D_t$  ( $1 < t < K$ ) that can be jointly encoded with  $H_t$  to form a sequence of bits having a total length equal to the height of the encoding tree.  $D_t$  is a multiplexing capacity of a given sequence of symbols. Calculation of the sum of the set of  $D_t$  associated with the symbol sequences forming the source  $H$  makes it possible to determine the multiplexing capacity of the flow  $S_H$ .

In a general manner, an arithmetic encoder or an encoding tree can be used to establish a code word prefix for each sequence of symbols, which is equivalent to defining several possible code words for certain symbol sequences. The choice between these possible code words will be made once the binary code to determine the code word suffix has been read and the code word has been formed by concatenation of the prefix and suffix.

Thus, the invention allows multiplexing of two data flows  $S_H$  and  $S_L$ , in order to reduce the error sensitivity of one of them  $S_H$ , designated as more important or higher priority. These two flows can be differentiated in the same signal source, in particular as in the following examples of  $S_H$  and  $S_L$  sources :

- low frequencies and high frequencies extracted by multi-resolution decomposition (filter banks, wavelet transforms) of a signal,
- texture information (e.g. DCT coefficients, wavelet coefficients) and movement information,

- most significant bits and least significant bits of wavelet coefficients or quantified samples of a signal.

Of course, the above enumeration is in no way exhaustive.

5

Furthermore, in that the code words are of fixed length (or if synchronization markers are used), the invention can be used to create a multiplexed code capable of jointly describing two flows, at least one of which has the benefit of perfect synchronization.

Appendix 1 - Formulas

$$(E1) \quad \mathcal{A} = \{a_1, \dots, a_i, \dots, a_\Omega\}$$

$$(E2) \quad \mu_i = P(a_i)$$

$$(E3) \quad \bar{\mu}_i = \frac{2^c}{2^c - \Omega_M} \mu_i$$

$$(E4) \quad N_i = (2^c - \Omega_m) \bar{\mu}_i$$

$$(E5) \quad \sum_{i \in \mathcal{A}} N_i = 2^c$$

$$(E6) \quad c_i \rightleftharpoons s_i, z_i$$

$$(E7) \quad \Lambda = \prod_{t=1}^{K_H} n_t$$

$$E8 \quad K'_t = \lceil \log_2 \Lambda_t \rceil$$

$$E9 \quad \gamma = \sum_{r=1}^{K_H} b_{r+K_B-Y_B} 2^{r-1}$$

$$0) \quad v_T \text{ bits} \rightleftharpoons \begin{cases} v_T, \text{ 2-valued variables} & e_{2,1}, \dots, e_{v_T} \\ v_T, \text{ 3-valued variables} & e_{1,2}, e_{2,2}, \dots, e_{v_T} \\ \dots & \dots \\ v_T, \text{ fvi-valued variables} & e_{1,v}, e_{2,v}, \dots, e_{v_T} \end{cases}$$

$$(E1) \quad \Omega \leq N \leq 2^L$$

$$E1 \quad D = \log_2 I$$

$$E.3) \quad P(S_t, S_{t+C}) = \frac{P(S_t) P(S_{t+C} | S_t)}{P(S_t)} \quad (S_t \in \mathcal{S})$$

**Appendix 2 - Algorithms**

$\gamma' = \gamma$   
 For  $t = 1 : K_H$   
 A1      $q_t = \gamma' \text{ modulo } n_t$   
         $\gamma' = \frac{\gamma' - q_t}{n_t}$   
 End for

$z = 0$   
 % while fj-valued variables remain to be obtained  
 While sum  $(d_j) > 0$   
 A2     % Calculate the number of times that the transformation  $T_z$  is used  
         $g_{T_z} = \text{floor}(\min(\frac{d_j}{v_{T_z,j}})) \quad v_{T_z,j} \neq 0$   
        % Calculate the number of  $f_j$ -valued variables  
        % which have not been transformed by the transformation  $T_z$   
        For each  $j$  between 1 and  $v$   
             $d_j = d_j - g_{T_z} * v_{T_z,j}$   
            %  
             $z = z + 1$

$e' = e$   
 For  $j = 1 : v$   
    For  $r = 1 : v_{T_z,j}$   
 A3      $e_{r,j} = e' \text{ modulo } f_j$   
         $e' = \frac{e' - e_{r,j}}{f_j}$   
    End for  
 End for



*For*  $j = 1 : \nu$

$t_j = 1$

*End for*

*For*  $t = 1 : K_H$

$q_t = 0$

A4

*For*  $j = \nu : 1$  by -1

*For*  $r = 1 : \alpha_{t,j}$

$q_t = q_t * f_j + F_j(t_j)$

$t_j = t_j +$

*End for*

*End for*

*End for* .